Overview of Software and Simulations

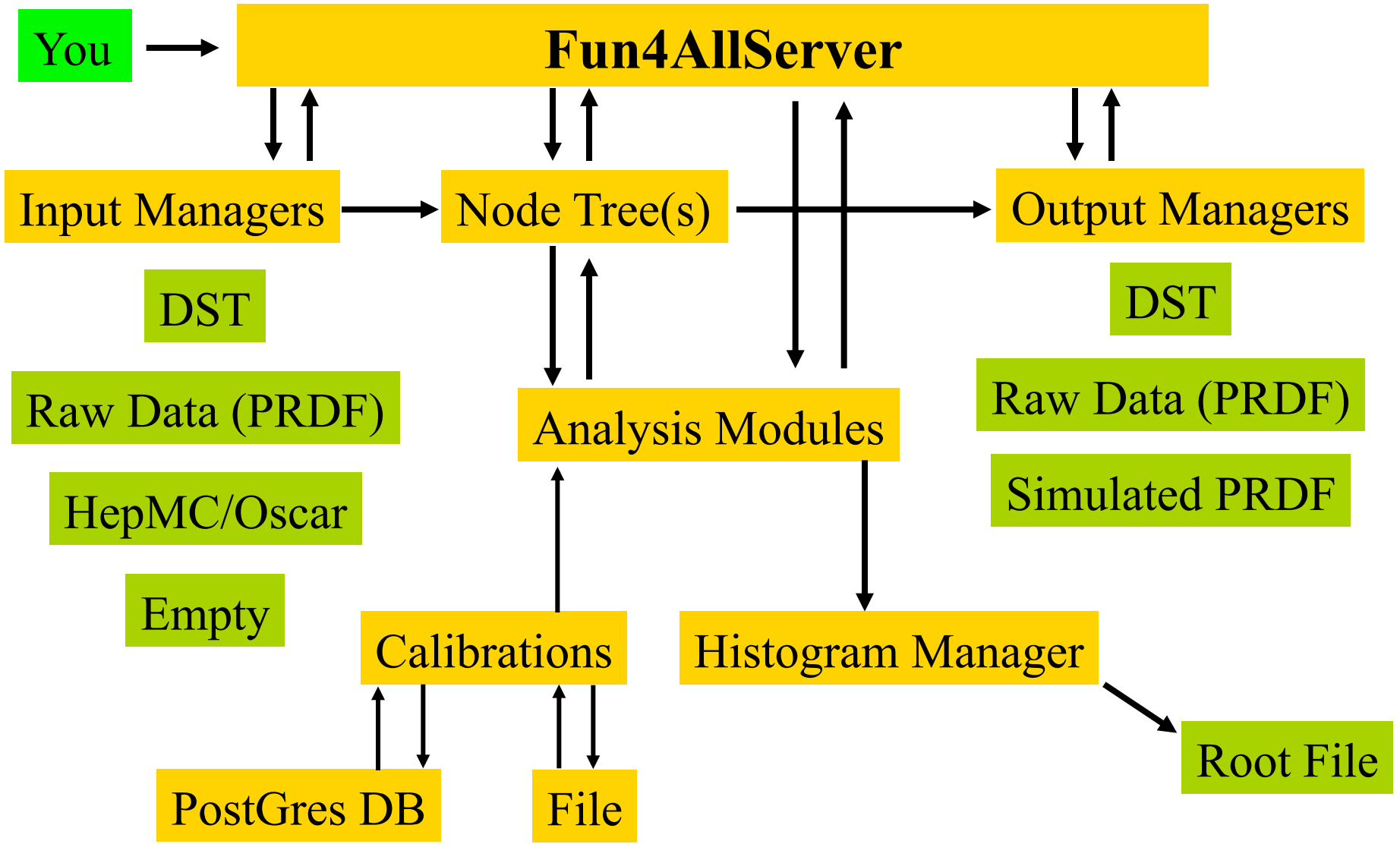
sFun4All

PHENIX framework history

- Development started in 2002, in use by PHENIX from 2003 on for reconstruction of real and simulated data, embedding and analysis
- Needed to get many subsystems who developed their code independently and with no real coordination under one umbrella
- Development driven by reconstruction and analysis needs (plus the urgent need to process incoming data)
- KISS + Modularity key to be able to evolve and adapt
- Configured by simple Root macros

It's a mature product, advantage that it saw data (unlike starting from scratch where you have sims for a long time)

Structure of Fun4All



The Node Tree

- The Node Tree is at the center of the Phenix software universe (but it's more or less invisible to you). It's the way we organize our data.
- **It is NOT a Root TTree**
- We have 3 different Types of Nodes:
  - PHCompositeNode: contains other Nodes
  - PHDataNode: contains any object
  - PHIODataNode: contains objects which can be written out to DST
- PHCompositeNodes and PHIODataNodes can be saved to a DST and read back
- This DST contains Root TTrees, the node structure is saved in the branch names. Due to Roots limitations not all objects can become PHIODataNodes (e.g. anything containing BOOST or G4).
- We currently save 2 Root TTrees in each output file, one contains the eventwise information, the other the runwise information
- Input Managers put objects as PHIODataNodes on the node tree, output managers save selected PHIODataNodes to a file.
- Fun4All can manage multiple independent node trees

Node Tree for sPHENIX

Print it from the cmd line with
Fun4AllServer *se = Fun4AllServer::instance();
se->Print("NODETREE");

```
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
  DST (PHCompositeNode)/
    HCALIN (PHCompositeNode)/
      G4HIT_HCALIN (PHIODataNode)
      G4HIT_ABSORBER_HCALIN (PHIODataNod
    SVTX (PHCompositeNode)/
      SvtxHitMap (PHIODataNode)
      SvtxClusterMap (PHIODataNode)
    SVTX_EVAL (PHCompositeNode)/
      SvtxClusterMap_G4HIT_SVTX_Links (PHIODataNode)
RUN (PHCompositeNode)/
    CYLINDERGEOM_SVTX (PHIODataNode)
    CYLINDERGEOM_SVTXSUPPORT (PHIODataNode)
    CYLINDERGEOM_EMCELECTRONICS_0 (PHIODataNode)
    CYLINDERGEOM_HCALIN_SPT (PHIODataNode)
PAR (PHCompositeNode)/
    SVTX (PHCompositeNode)/
      SvtxBeamSpot (PHIODataNode)
```

TOP: Top of Default Node Tree, creation and populating of other node trees is possible (used for embedding)

Node Tree for sPHENIX

Print it from the cmd line with
Fun4AllServer *se = Fun4AllServer::instance();
se->Print("NODETREE");

```
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
  DST (PHCompositeNode)/
    HCALIN (PHCompositeNode)/
      G4HIT_HCALIN (PHIODataNode)
      G4HIT_ABSORBER_HCALIN (PHIODataNode)
    SVTX (PHCompositeNode)/
      SvtxHitMap (PHIODataNode)
      SvtxClusterMap (PHIODataNode)
    SVTX_EVAL (PHCompositeNode)/
      SvtxClusterMap_G4HIT_SVTX_Links (PHIODataNo
  RUN (PHCompositeNode)/
    CYLINDERGEOM_SVTX (PHIODataNode)
    CYLINDERGEOM_SVTXSUPPORT (PHIODataNode)
    CYLINDERGEOM_EMCELECTRONICS_0 (PHIODa
    CYLINDERGEOM_HCALIN_SPT (PHIODataNode)
PAR (PHCompositeNode)/
    SVTX (PHCompositeNode)/
      SvtxBeamSpot (PHIODataNode)
```

DST and RUN Node: default for I/O

- DST – eventwise
- RUN - runwise

Objects under the DST node are reset after every event to prevent event mixing. You can select the objects to be saved in the output file. Subnodes like SVTX are saved and restored as well. DST/RUN nodes can be restored from file under other TopNodes
ROOT restrictions apply:
Objects cannot be added while running to avoid event mixing

Node Tree for sPHENIX

Print it from the cmd line with
Fun4AllServer *se = Fun4AllServer::instance();
se->Print("NODETREE");

```
Node Tree under TopNode TOP
TOP (PHCompositeNode)/
  DST (PHCompositeNode)/
    HCALIN (PHCompositeNode)/
      G4HIT_HCALIN (PHIODataNode)
      G4HIT_ABSORBER_HCALIN (PHIODataN
    SVTX (PHCompositeNode)/
      SvtxHitMap (PHIODataNode)
      SvtxClusterMap (PHIODataNode)
    SVTX_EVAL (PHCompositeNode)/
      SvtxClusterMap_G4HIT_SVTX_Links (PHIODataNode)
RUN (PHCompositeNode)/
    CYLINDERGEOM_SVTX (PHIODataNode)
    CYLINDERGEOM_SVTXSUPPORT (PHIODataNode)
    CYLINDERGEOM_EMCELECTRONICS_0 (PHIODataNode)
    CYLINDERGEOM_HCALIN_SPT (PHIODataNode)
PAR (PHCompositeNode)/
    SVTX (PHCompositeNode)/
      SvtxBeamSpot (PHIODataNode)
```

Users can add their own branches.
Resetting the objects (if needed)
is their responsibility.

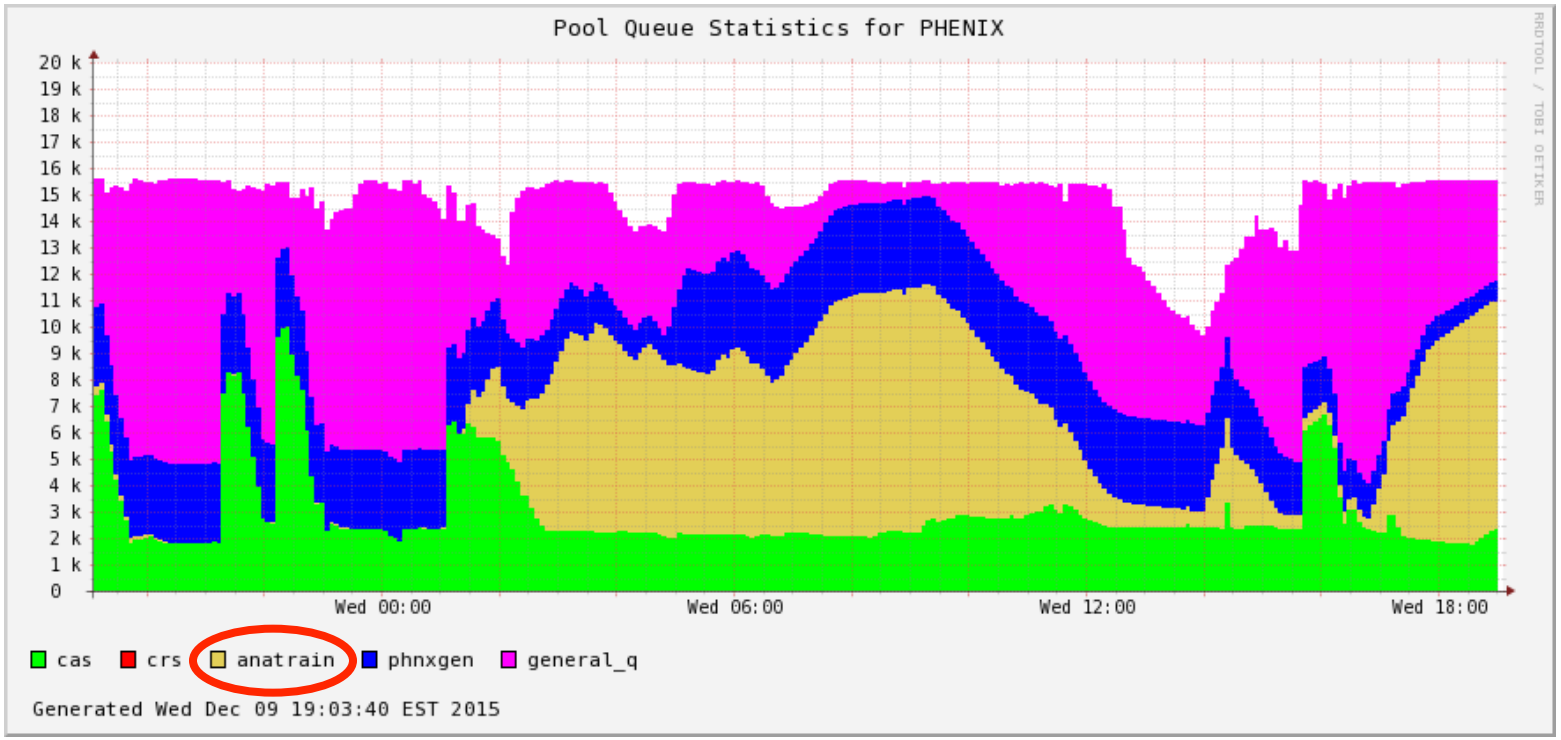
Keep it simple - Analysis Modules

You need to inherit from the SubsysReco Baseclass (offline/framework/fun4all/SubsysReco.h) which gives the methods which are called by Fun4All.
If you don't implement all of them it's perfectly fine

- Init(PHCompositeNode *topNode): called once when you register the module with the Fun4AllServer
- InitRun(PHCompositeNode *topNode): called whenever data from a new run is encountered
- Process_event (PHCompositeNode *topNode): called for every event
- ResetEvent(PHCompositeNode *topNode): called after each event is processed so you can clean up leftovers of this event in your code
- EndRun(const int runnumber): called before the InitRun is called (caveat the Node tree already contains the data from the first event of the new run)
- End(PHCompositeNode *topNode): Last call before we quit

If you create another node tree you can tell Fun4All to call your module with the respective topNode when you register your module

The PHENIX Analysis Taxi



This approach enabled us to develop a system (formerly Analysis Train but now for single modules) to run modules centrally on demand relieving the users from dealing with all those batch failure modes and making sure they got all files. All datasets since 2003 are online available.

Scheme can be adapted to read sPHENIX simulations

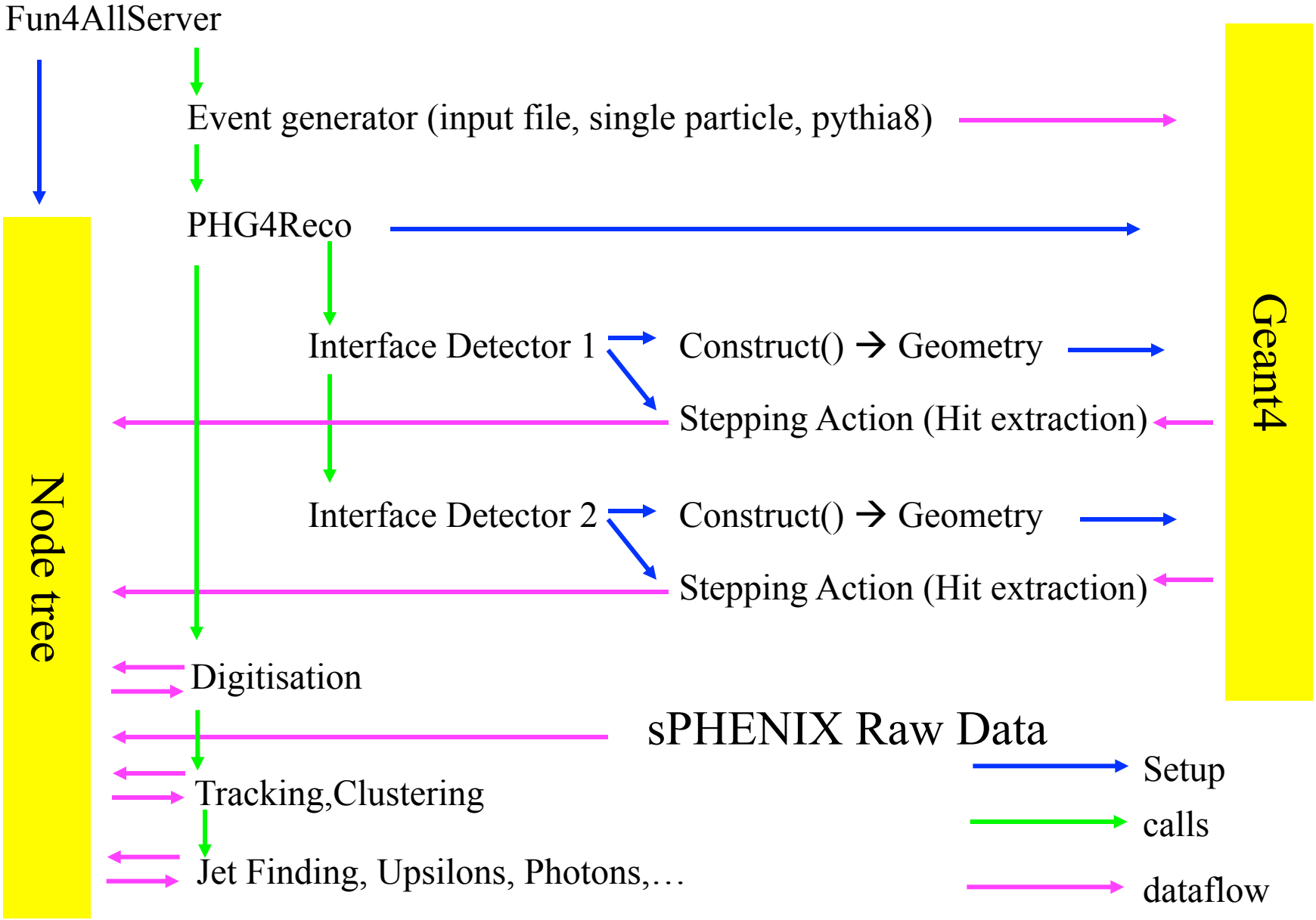
What to take from here

- Fun4All is a well developed mature framework but not overcomplicated, features are driven by real processing and analysis needs
- Standard C++, Root with shared libraries, configured and run by CINT macros
- We try to stay away from Root, only used if it is the best/only solution
- Writing multiple parallel streams is supported, events destined for any output stream can be selected by modules
- Synchronized parallel reading of input files, no need to have all objects you want in single file
- We have a Calibration Database scheme
- Users have to write analysis code in C++

sPHENIX code history

- Effort started 4 years ago, the decision was to go with G4 (hadron calorimeters) and use Fun4All as framework so all development could concentrate on G4
- The G4 simulation engine is implemented as an analysis module, the G4 command line interface is still intact and can be called from the Root prompt
- Generic cylinders, boxes and cones available if you want to try something quick, a "black hole" provides leakage detections
- Truth information is propagated for evaluation
- Higher level geometries: spacal (1d/2d projective), inner and outer hcal with tilted slats, svtx ladders
- Modular simulation setup – sPHENIX components (SVTX, EMC, Hcals, passive materials) are put together and configured in root macros.
- Code: https://github.com/sPHENIX-Collaboration/coresoftware
- Used to analyze upcoming Test Beam Data, same framework for real data and simulations
- Lessons learned from PHENIX are being applied

G4 program flow in sPHENIX

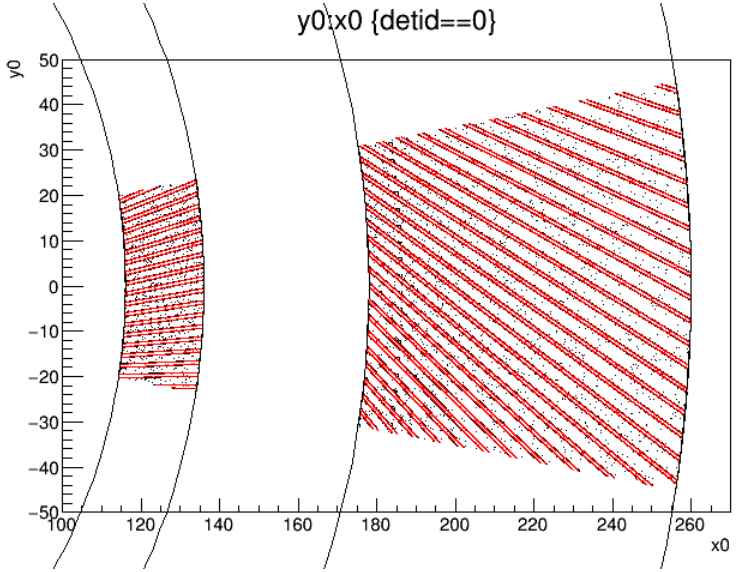


Example Hcal setup macro

```
PHG4InnerHcalSubsystem *hcal;
hcal = new PHG4InnerHcalSubsystem("HCALIN");
hcal->set_string_param("material",“SS310");
hcal->set_int_param("ncross",7);
hcal->set_int_param("n_scinti_plates",331);
hcal->set_int_param("n_scinti_tiles",11);
hcal->set_int_param("light_scint_model",0);
hcal->set_double_param("scinti_tile_thickness",0.6);
hcal->SetActive();
hcal->SuperDetector("HCALIN");
if (absorberactive)  hcal->SetAbsorberActive();
hcal->OverlapCheck(0);
g4Reco->registerSubsystem( hcal );
```

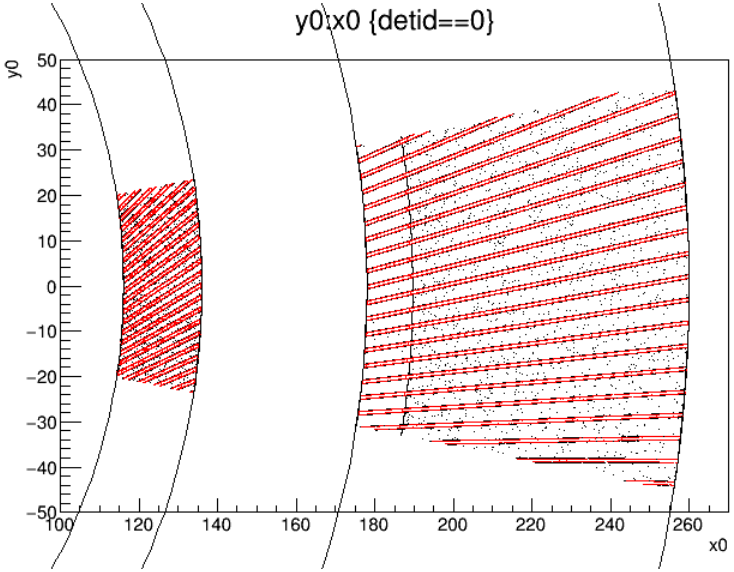
Parameter space scanning made simple
(just to mention this, the actual code in the detector construction is a **LOT** more complex)

Hcal parameter space scan

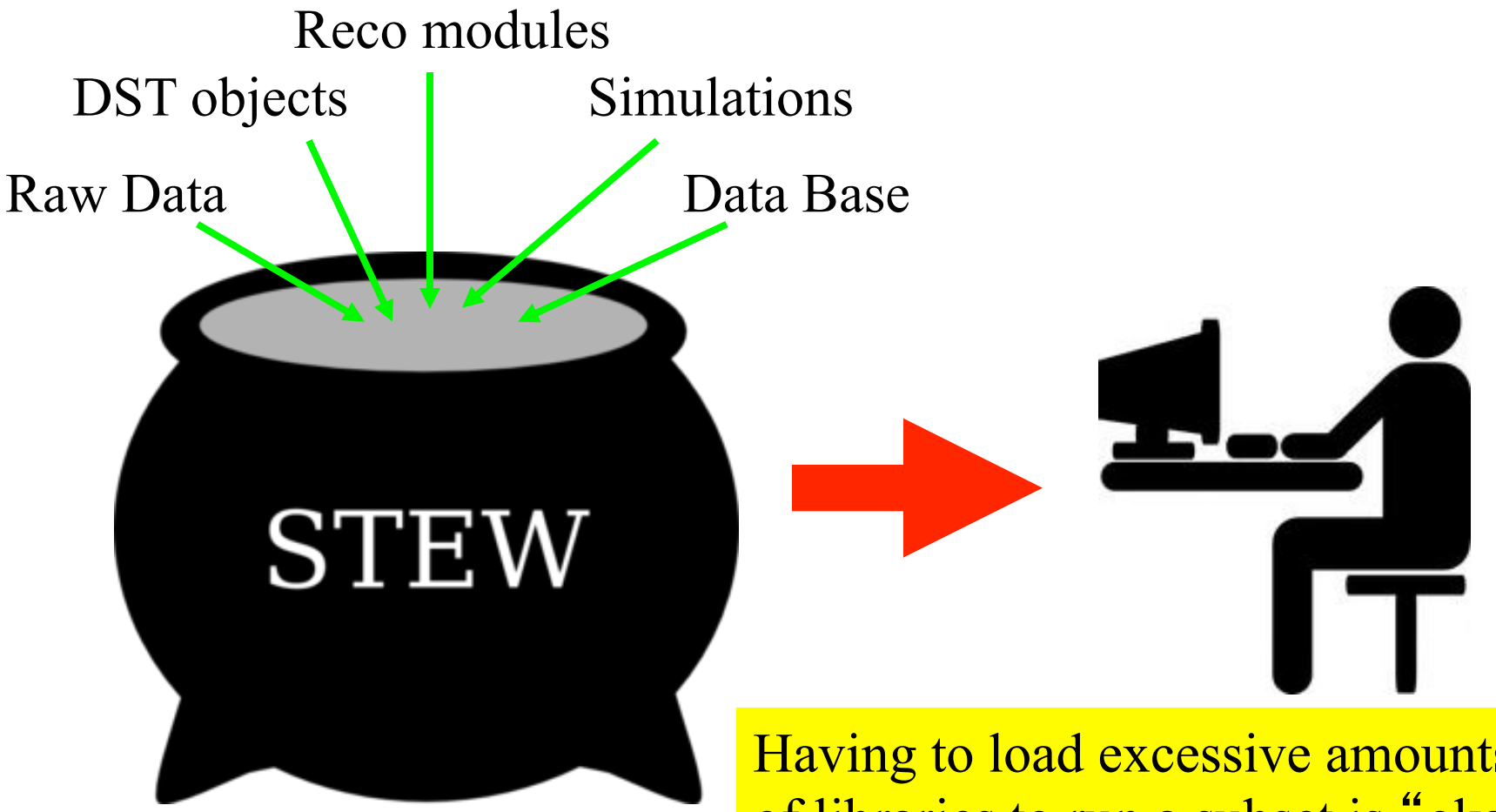


Geantino Scan in $\pm 10^0$
for different tilt angles

To help us to give an answer to the question what effects the chosen tilt angle has on our physics



Lesson learned: Avoid library Interdependencies



Having to load excessive amounts of libraries to run a subset is “okay” on rcf with afs client caching but a deal breaker on the grid

Revisited sPHENIX library dependencies: Only load what you eat

Root based DST analysis

DST analysis with Fun4All

sPHENIX Simulations

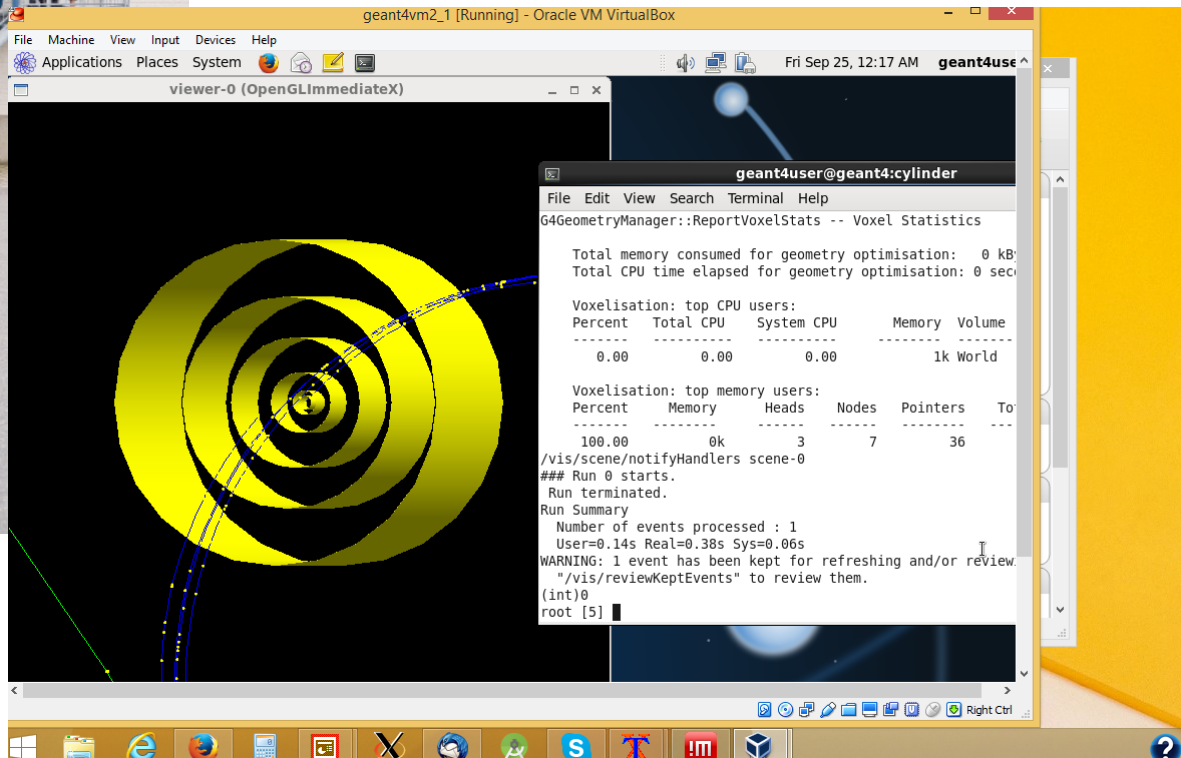
Extras:
DB access
Raw Data (lots of fiber)

I/O libraries, framework, simulations, raw data handling and DB libraries cleanly separated → significantly fewer bytes need to be transported
→ we can run simulations easily on the OSG

sPHENIX Virtual machine

Before everyone queues up for an rcf account

Try our virtual machine (Thanks to Martin)



Status and Future plans

- We have a working software framework based on 10+ years of experience with PHENIX
- Our simulation setup is modular and easily configurable
- To get new users up to speed – rather than extensive documentation - keep examples, tutorials and cut and paste starters up to date

Backup

Here a more scary geometry

```
G4VSolid*
PHG4OuterHcalDetector::ConstructSteelPlate(G4LogicalVolume* hcalenvelope)
{
  // calculate steel plate on top of the scinti box. Lower edge is the upper edge of
  // the scintibox + 1/2 the airgap
  double mid_radius = params->inner_radius + (params->outer_radius - params->inner_radius) / 2.;
  // first the lower edge, just like the scinti box, just add the air gap
  // and calculate intersection of edge with inner and outer radius.
  Point_2 p_in_1(mid_radius, 0); // center of lower scintillator
  double angle_mid_scinti = M_PI / 2. + params->tilt_angle / rad;
  double xcoord = params->scinti_gap / 2. * cos(angle_mid_scinti / rad) + mid_radius;
  double ycoord =   params->scinti_gap / 2. * sin(angle_mid_scinti / rad) + 0;
  Point_2 p_loweredge(xcoord, ycoord);
  Line_2 s2(p_in_1, p_loweredge); // center vertical
  Line_2 perp =  s2.perpendicular(p_loweredge); // that is the lower edge of the steel plate
  Point_2 sc1(params->inner_radius, 0), sc2(0, params->inner_radius), sc3(-params->inner_radius, 0);
  Circle_2 inner_circle(sc1, sc2, sc3);
  vector< CGAL::Object > res;
  CGAL::intersection(inner_circle, perp, std::back_inserter(res));
  Point_2 lowerleft;
  vector< CGAL::Object >::const_iterator iter;
  for (iter = res.begin(); iter != res.end(); ++iter)
    {
     CGAL::Object obj = *iter;
     if (const std::pair<CGAL::Circular_arc_point_2<Circular_k>, unsigned> *point = CGAL::object_cast<std::pair<CGAL::Circular_arc_point_2<C
                 {
                  if (CGAL::to_double(point->first.x()) > 0)
                    {
                     Point_2 pntmp(CGAL::to_double(point->first.x()), CGAL::to_double(point->first.y()));
                     lowerleft = pntmp;
                    }
                 }
```

Here a more scary geometry

```
                }
    else
                {
                  cout << "CGAL::Object type not pair..." << endl;
                }
  }
Point_2 so1(params->outer_radius, 0), so2(0, params->outer_radius), so3(-params->outer_radius, 0);
Circle_2 outer_circle(so1, so2, so3);
res.clear(); // just clear the content from the last intersection search
CGAL::intersection(outer_circle, perp, std::back_inserter(res));
Point_2 lowerright;
for (iter = res.begin(); iter != res.end(); ++iter)
  {
    CGAL::Object obj = *iter;
    if (const std::pair<CGAL::Circular_arc_point_2<Circular_k>, unsigned> *point = CGAL::object_cast<std::pair<CGAL::Circular_arc_point_2<C
                {
                  if (CGAL::to_double(point->first.x()) >  CGAL::to_double(p_loweredge.x()))
                    {
                      Point_2 pntmp(CGAL::to_double(point->first.x()), CGAL::to_double(point->first.y()));
                      lowerright = pntmp;
                    }
                }
    else
                {
                  cout << "CGAL::Object type not pair..." << endl;
                }
  }
// now we have the lower left and rigth corner, now find the upper edge
// find the center of the upper scintilator
```

Here a more scary geometry

```
double phi_midpoint = 2 * M_PI / params->n_scinti_plates;
  double xmidpoint = cos(phi_midpoint) * mid_radius;
  double ymidpoint = sin(phi_midpoint) * mid_radius;
  // angle of perp line at center of scintillator
  angle_mid_scinti = (M_PI / 2. - phi_midpoint) - (M_PI / 2. + params->tilt_angle / rad);
  double xcoordup = xmidpoint - params->scinti_gap / 2. * sin(angle_mid_scinti / rad);
  double ycoordup = ymidpoint - params->scinti_gap / 2. * cos(angle_mid_scinti / rad);
  Point_2 upperleft;
  Point_2 upperright;
  Point_2 mid_upperscint(xmidpoint, ymidpoint);
  Point_2 p_upperedge(xcoordup, ycoordup);
  {
    Line_2 sup(mid_upperscint, p_upperedge); // center vertical
    Line_2 perp =  sup.perpendicular(p_upperedge); // that is the upper edge of the steel plate
    Point_2 sc1(params->inner_radius, 0), sc2(0, params->inner_radius), sc3(-params->inner_radius, 0);
    Circle_2 inner_circle(sc1, sc2, sc3);
    vector< CGAL::Object > res;
    CGAL::intersection(inner_circle, perp, std::back_inserter(res));
    vector< CGAL::Object >::const_iterator iter;
    double pxmax = 0.;
    for (iter = res.begin(); iter != res.end(); ++iter)
      {
                CGAL::Object obj = *iter;
                if (const std::pair<CGAL::Circular_arc_point_2<Circular_k>, unsigned> *point = CGAL::object_cast<std::pair<CGAL::Circular_arc
                  {
                   if (CGAL::to_double(point->first.x()) > pxmax)
                     {
                                pxmax = CGAL::to_double(point->first.x());
                                Point_2 pntmp(CGAL::to_double(point->first.x()), CGAL::to_double(point->first.y()));
                                upperleft = pntmp;
                     }
```

Here a more scary geometry

```
                else
                  {
                   cout << "CGAL::Object type not pair..." << endl;
                  }
    }
  Point_2 so1(params->outer_radius, 0), so2(0, params->outer_radius), so3(-params->outer_radius, 0);
  Circle_2 outer_circle(so1, so2, so3);
  res.clear(); // just clear the content from the last intersection search
  CGAL::intersection(outer_circle, perp, std::back_inserter(res));
  for (iter = res.begin(); iter != res.end(); ++iter)
    {
                CGAL::Object obj = *iter;
                if (const std::pair<CGAL::Circular_arc_point_2<Circular_k>, unsigned> *point = CGAL::object_cast<std::pair<CGAL::Circular_arc
                  {
                   if (CGAL::to_double(point->first.x()) >  CGAL::to_double(p_loweredge.x()))
                     {
                                Point_2 pntmp(CGAL::to_double(point->first.x()), CGAL::to_double(point->first.y()));
                                upperright = pntmp;
                     }
                  }
                else
                  {
                   cout << "CGAL::Object type not pair..." << endl;
                  }
    }
}
// the left corners are on a secant with the inner boundary, they need to be shifted
// to be a tangent at the center
ShiftSecantToTangent(lowerleft, upperleft, upperright, lowerright);
```

Here a more scary geometry

```
G4TwoVector v1(CGAL::to_double(upperleft.x()), CGAL::to_double(upperleft.y()));
 G4TwoVector v2(CGAL::to_double(upperright.x()), CGAL::to_double(upperright.y()));
 G4TwoVector v3(CGAL::to_double(lowerright.x()), CGAL::to_double(lowerright.y()));
 G4TwoVector v4(CGAL::to_double(lowerleft.x()), CGAL::to_double(lowerleft.y()));
 std::vector<G4TwoVector> vertexes;
 vertexes.push_back(v1);
 vertexes.push_back(v2);
 vertexes.push_back(v3);
 vertexes.push_back(v4);
 G4TwoVector zero(0, 0);
 G4VSolid* steel_plate_uncut =  new G4ExtrudedSolid("SteelPlateUnCut",
                                                    vertexes,
                                                    params->size_z  / 2.0,
                                                    zero, 1.0,
                                                    zero, 1.0);

    G4RotationMatrix *rotm = new G4RotationMatrix();
          rotm->rotateX(-90 * deg);

 // now cut out space for magnet at the ends
     G4VSolid* steel_firstcut_solid = new G4SubtractionSolid("SteelPlateFirstCut",steel_plate_uncut,steel_cutout_for_magnet,rotm,G4ThreeVector
     //   DisplayVolume(steel_plate_uncut, hcalenvelope);
   //    DisplayVolume(steel_cutout_for_magnet, hcalenvelope);
   //    DisplayVolume(steel_cutout_for_magnet, hcalenvelope,rotm);
     //    DisplayVolume(steel_firstcut_solid, hcalenvelope);
    rotm = new G4RotationMatrix();
          rotm->rotateX(90 * deg);
     G4VSolid* steel_cut_solid = new G4SubtractionSolid("SteelPlateCut",steel_firstcut_solid,steel_cutout_for_magnet,rotm,G4ThreeVector(0,0,0)
     //          DisplayVolume(steel_cut_solid, hcalenvelope);

 return steel_cut_solid;
```